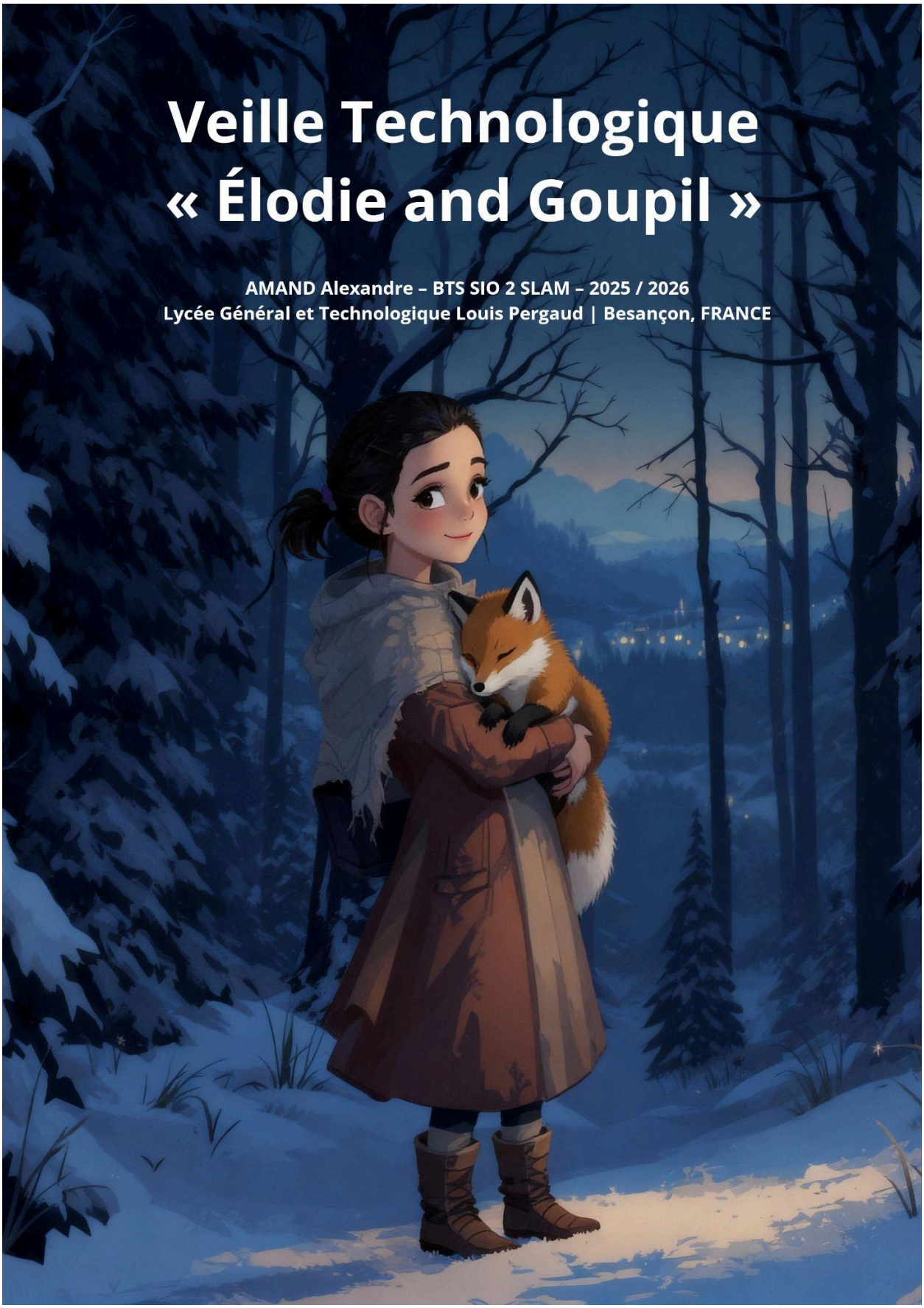
Veille Technologique
« Élodie and Goupil »

AMAND Alexandre – BTS SIO 2 SLAM – 2025 / 2026
Lycée Général et Technologique Louis Pergaud | Besançon, FRANCE



Tables des Matières :

0. Introduction à la Veille Technologique

Une **veille technologique** est un processus systématique de collecte, d'analyse et de diffusion d'informations sur les avancées techniques, les nouvelles technologies, et les meilleures pratiques dans un domaine spécifique.
L'objectif est d'aider à la prise de décision stratégique, d'anticiper les changements, d'identifier les opportunités et de réduire les risques techniques.

Ce rapport documente la veille technologique progressive effectuée pour le projet de jeu vidéo "Élodie et Goupil", retraçant la transition d'une architecture moteur de jeu traditionnelle (basée sur Quake) vers une pile technologique Web moderne, performante et résiliente (basée sur BabylonJS et les Web Workers).

Partie 1 : Projet "Élodie et Goupil" - Contexte et Ambition Narrative

Le projet "Élodie et Goupil" est un jeu vidéo 3D conçu comme une aventure narrative et émotionnelle, jouable en solo ou en coopération (écran scindé local et en ligne).
L'ambition narrative est centrale et guide l'ensemble des choix techniques.

1.1 Synopsis de l'Histoire

Inspirée des nouvelles "De Goupil à Margot" de Louis Pergaud, l'histoire se déroule dans une forêt européenne en hiver.
Le récit suit Élodie, une jeune fille de 12 ans, harcelée par les enfants du village (qui la traitent de "meurtrière") et vivant dans l'ombre d'un traumatisme familial.
Lors d'une dispute violente, son père bûcheron et alcoolisé a tenté d'agresser sa mère (la couturière) ; en voulant protéger cette dernière, Élodie a accidentellement tiré sur son père avec une arme de chasse, le tuant.

L'intrigue débute lorsque des chasseurs locaux, menés par l'antagoniste **Gustave**, capturent un jeune renardeau, **Goupil**, avec l'intention de l'empailler.
Ils forcent la mère d'Élodie, une couturière marginalisée (menacée de révéler son divorce passé, à l'époque considéré comme une honte), à accepter la tâche.

Élodie, voyant sa propre blessure et son impuissance dans celles de l'animal, décide de le sauver.
Avec l'aide de **Camille**, l'apothicaire du village, Goupil est soigné en secret.

Élodie prend alors la décision de fuir le village avec Goupil, marquant le début de leur voyage.

1.2 Déroulé Narratif et Gameplay

Le jeu est structuré en huit actes, retraçant la fuite d'Élodie et Goupil, leur survie dans la forêt hivernale, et leur confrontation avec les menaces humaines et naturelles.

- **Acte 1 (La Chute de Goupil) & Acte 2 (La Forêt des Premiers Pas) :** Ces actes couvrent la capture de Goupil, le flashback sur le traumatisme d'Élodie, la décision de fuir, et les premiers obstacles (traversée de rivière, évitement des patrouilles).
- **Actes 3 à 8 (Le Voyage, Les Secrets, La Résolution) :** Ces actes développent la survie, l'exploration de zones (village, forêt profonde, ruines, rivière gelée), et la confrontation avec les antagonistes.

Les mécaniques de jeu (Gameplay) et les systèmes (Systèmes) sont conçus pour renforcer l'immersion narrative :

- **Mécaniques de Survie :** Gestion de la **santé** et du **froid** (nécessitant repos et abris).
- **Artisanat (Crafting) :** Un système de crafting léger permet de fabriquer des abris, des feux, des bandages et des remèdes à partir de ressources naturelles (bois, herbes, tissus).
- **Lien Compagnon (IA de Goupil) :** Le cœur du gameplay repose sur le lien affectif entre les deux protagonistes. Le joueur doit soigner et interagir avec Goupil.
  La confiance du compagnon (jauge invisible) influence son comportement (alerte des dangers, aide à l'exploration, obéissance aux ordres).
- **Furtivité et Exploration :** Le joueur doit éviter les chasseurs (Gustave, Bastien) et certains animaux sauvages.
- **Narration Environnementale :** Des flashbacks, des objets-souvenirs et des rencontres (comme avec **Louise**) dévoilent le lore et le passé d'Élodie.
- **Fins Multiples :** L'histoire propose plusieurs fins (Exil, Ermite, Confrontation) basées sur les choix narratifs du joueur.

1.3 Personnages et Ambiance

Le récit est soutenu par des personnages secondaires clés, tels que **Camille** (l'apothicaire allié), **Louise** (la conteuse), **Gustave** (le chasseur antagoniste), **Bastien** (le harceleur), **Victorine** (la fille du chasseur) et **Le Maire** (figure d'autorité neutre).

L'ambiance recherchée est poétique, mélancolique et immersive, similaire à des jeux comme *Never Alone*, *Endling*, *The Last Guardian* ou *Spirit of the North*.

Partie 2 : L'Hypothèse Initiale - La Technologie Quake (L'impasse)

La première phase de la veille technologique du projet a consisté à évaluer la pertinence du moteur Quake, en se basant sur l'étude de cas du projet open-source **Nazi Zombies: Portable (NZP)**.

2.1 Architecture et Avantages Identifiés (NZP)

L'architecture de NZP semblait initialement prometteuse.
Elle est décomposée en plusieurs parties, incluant différents forks du moteur Quake adaptés à chaque plateforme :

- `vril-engine` (PSP, 3DS)
- `fteqw` (PC, Web)
- `glquake` (3DS)
- `quakespasm` (Switch, Vita)

Les avantages identifiés étaient significatifs :

- **Multiplateforme :** Une couverture exceptionnelle (PC, Web, consoles portables).
- **Structure Standardisée :** Une séparation claire entre les assets, le code de jeu (`quakec`) et les outils.
- **Fonctionnalités Moteur :** Support natif de l'écran scindé (split-screen) et un éditeur de niveau établi (TrenchBoom).

2.2 Analyse Technique Approfondie du Moteur Quake (NZP)

Une analyse approfondie du pipeline de build de NZP a été menée pour comprendre son fonctionnement interne.

- **Système de Build :** Le build repose sur des scripts shell (`qc-compiler-gnu.sh`) qui utilisent le compilateur `fteqcc`.
  Le projet utilise une structure modulaire séparant le code serveur (SSQC), client (CSQC) et menu (MenuQC), chacun ayant ses propres `defs.qc` (fichiers de définitions).
- **Gestion des Assets (PAK/WAD) :** L'analyse a couvert la manière dont Quake gère les assets, en étudiant la création de fichiers `.pak` (archives de jeu) et de fichiers `.wad` (textures).
  Des scripts DenoJS ont été développés pour automatiser ce pipeline de build Quake, montrant une première transition vers des outils modernes.
- **Couche de Compatibilité (CSV) :** L'étude du script `qc_hash_generator.py` a révélé l'utilisation d'un `asset_conversion_table.csv`. Il s'agit d'une couche de compatibilité historique.
  Elle sert à mapper les anciens chemins d'assets de Quake (ex: `progs/player.mdl`) vers les nouveaux chemins utilisés dans NZP (ex: `models/player.mdl`), afin de ne

pas avoir à réécrire le code QuakeC hérité.
Si les références dans le code QuakeC sont correctes dès le départ, ce mapping devient inutile.

2.3 La Limitation Critique (UI) et l'Abandon de Quake

L'investigation technique a révélé une limitation rédhibitoire pour l'utilisation des technologies Quake pour le projet "Élodie et Goupil" :

1. **Absence de CSQC :** L'analyse des logs et du code a montré que le moteur NZP n'appelle *pas* les fonctions standards `CSQC_Init` ou `CSQC_InputEvent` (le code client standard de Quake).
2. **Remplacement par SUI :** NZP utilise une bibliothèque d'interface utilisateur personnalisée, `sui_sys.qc` (Shuld's Simple UI), qui est appelée directement depuis le code côté serveur (SSQC).
3. **L'Impasse (Code Moteur C++) :** La découverte critique est que, à l'exception de la version `fteqw` (PC/Web), la logique graphique et l'interface utilisateur (UI) de NZP sont codées en dur **directement dans les moteurs C/C++** de chaque fork (Vril, Quakespasm, GLQuake).

Cette approche signifie que pour implémenter l'interface de crafting, de narration et de gestion du compagnon d'Élodie, il aurait été nécessaire de **répliquer et d'adapter la partie graphique plusieurs fois**, dans chaque moteur différents.

Cet effort de maintenance étant jugé rédhibitoire et contraire à une logique de développement modulaire, la technologie Quake a été abandonnée.

Partie 3 : Le Pivot Stratégique - Les Technologies Web et BabylonJS

Face à l'impasse du moteur Quake, la veille s'est réorientée vers les technologies Web modernes (HTML5, JS/TS, WebGL/WebGPU) et le moteur 3D **BabylonJS**, jugées plus adaptés à la création d'interfaces interactives et à un développement multiplateforme unifié.

3.1 Architecture Moteur (BabylonJS)

Une nouvelle architecture a été conçue, basée sur BabylonJS, en séparant les responsabilités via des **Web Workers** pour garantir qu'aucune logique lourde ne bloque le thread principal (rendu).

- **Thread Principal (`Main`) :** Gère uniquement le rendu BabylonJS (WebGL/WebGPU) et la capture des entrées utilisateur. Il ne doit jamais être bloqué.
- **Worker Client (`ClientWorker`) :** S'exécute dans un thread séparé. Gère la logique spécifique au joueur, les intentions (commandes), et la communication.

- **Worker Serveur (`ServerWorker`) :** Simule l'intégralité du monde de jeu (physique, IA, état).
  C'est un worker hybride :
  - **Mode Hors-Ligne (Fallback) :** Il s'exécute localement dans le navigateur et agit comme un **Serveur Local**.
  - **Mode En-Ligne :** Il agit comme un **Proxy de Communication** qui relaie les messages entre le `ClientWorker` et le Serveur Distant (WebSocket/WebRTC).
- **Gestionnaire (`Manager`) :** Orchestre la configuration et la communication.

3.2 Communication "Pure" et Résilience Réseau

Pour optimiser la performance et respecter l'isolation des threads, l'architecture utilise des API web avancées :

- **MessageChannel :** Pour établir une communication "pure" et directe (sans copie de mémoire si possible) entre le `ClientWorker` et le `ServerWorker` (local) sans jamais passer par le thread principal, évitant ainsi tout goulot d'étranglement.
- **Résilience Réseau :** En mode en ligne, si la connexion au serveur distant (WebSocket/WebRTC) est perdue, le client stocke les actions du joueur dans un tampon.
- **Fallback Automatique :** Le `ServerWorker` est capable de basculer de manière transparente d'un transport distant à un transport local sans perte d'états en cas de déconnexion définitive. Cette option est contrôlable par le joueur.
- **Transition Chaude (Hot Swap) :** Pour une transition sans couture, le `ServerWorker` utilise le `lastKnownRemoteState` (dernier état reçu du serveur distant) pour initialiser la simulation locale, permettant au joueur de continuer sa partie sans interruption.
- **Gestion des Intentions :** Un `ClientWorker` stocke les actions du joueur durant une déconnexion temporaire et les renvoie au serveur distant lors de la reconnexion pour resynchronisation.

3.3 Modularité (Multi-Repo) et Coopération Hybride

L'architecture est conçue en **multi-repo** pour une séparation stricte des responsabilités :

- **`Elodie-Goupil` :** Le dépôt orchestrateur (méta-repo).
- **`engine`/`shared` :** Contient tout le code réutilisable (ECS, rendu, système de fallback, réseau).
- **`client`/`server` :** Contient uniquement la logique métier du jeu (scripts d'IA, règles de gameplay, UI).
- **`assets` :** Dépôt contenant les sources des assets catégorisées.
- **`web-server` :** Serveur Distant permettant le Multijoueur En-Ligne.
- **`native`/`react-native` :** Dépôts de Compilation pour différentes plateformes.

Cette structure supporte la **Coopération Hybride** :

- **Co-op Locale (Écran Scindé) :** Gérée par des modules génériques disponible dans le code réutilisable.
- **Co-op en Ligne :** Gérée par le `ServerWorker` en mode proxy.

3.4 Pérennité (Préparation SSR)

La structure inclut des abstractions pour le rendu et des placeholders pour un futur **Rendu Côté Serveur (SSR)**, permettant d'envisager un streaming du jeu (type cloud gaming) si la machine client n'est pas assez puissante.

Partie 4 : Pipeline d'Assets Optimisé pour le Web (BabylonJS)

La transition vers BabylonJS s'accompagne d'une refonte complète du pipeline d'assets, s'éloignant des formats propriétaires (PAK, WAD) pour adopter des standards ouverts, libres de droits et optimisés pour le GPU.

4.1 Formats de Scène et Logique (glTF 2.0)

- **Format :** Le standard est le **glTF 2.0** (binaire `.glb`).
- **Logique Map-like :** Pour répliquer la flexibilité des entités Quake, la logique de jeu (ex: `classname: "func_door"`, `targetname: "door1"`) est encodée dans le champ **`extras`** du glTF.
- **Workflow :** Le pipeline de production passe par Blender.
  Les artistes ajoutent des "Propriétés Personnalisées" (Custom Properties) qui sont exportées dans `extras`.
  BabylonJS lit ensuite ces `extras` via l'évènement `OnPluginActivatedObservable` pour instancier les entités ECS correspondantes.
- **Modularité :** Le système différencie les assets statiques (intégrés au `.glb` de la carte) des assets dynamiques (référencés par `modelpath` dans les `extras` et instanciés par le moteur), optimisant ainsi la mémoire.

4.2 Textures (KTX2)

- **Format :** Pour une performance GPU maximale, toutes les textures 3D (Albedo, Normal, PBR) utilisent le format **KTX2 (Basis Universal)**.
- **Justification :** Ce format permet une compression GPU-native.
  Le moteur n'a pas besoin de décompresser le PNG/JPEG côté CPU, la texture est transcodée à la volée par le pilote graphique (en ASTC, BCn, ETCn...) selon la plateforme.
- **Veille Alpha :** L'analyse des options (ETC1S vs UASTC) a déterminé que le mode **UASTC** est le meilleur compromis, offrant une qualité visuelle quasi-sans perte et un support complet du canal alpha, ce qui est crucial pour le projet.

4.3 Interface (SVG, Lottie, WebP)

L'interface utilisateur (UI) est construite avec des formats web modernes pour garantir la légèreté et la flexibilité, en remplacement du terme "GFX" de Quake :

- **`ui/svg/` :** Icônes vectorielles (résolution infinie)
- **`ui/lottie/` :** Fichiers `.json` Lottie pour les animations 2D complexes (HUD, menus)
- **`ui/raster/` :** Fichiers `.webp` (ou AVIF) pour les images UI bitmap (sprites, fonds)

4.4 Audio, Vidéo et Données

- **Audio :** La musique et les voix sont standardisées en **WebM (Opus)**.
  Les effets sonores courts (FX) ou bouclant précisément utilisent **Ogg (Vorbis)**.

- **Vidéo :** Les cinématiques utilisent **WebM (VP9 + Opus)** (ou AV1), pour le meilleur ratio qualité/poids sur le web.

- **Données ("Meta-Assets") :** Le dossier `data/` contient les définitions non-visuelles :

  - `data/configs/` : Paramètres globaux.
  - `data/entities/` : Définitions des `classname` (comportements, stats).
  - `data/localisations/` : Fichiers de localisation (traduction).
  - `data/physics/` : Fichiers `.bin` (ou `.json`) pour les données de physique (ex: NavMesh pour l'IA Havok).

- **Shaders :** Les shaders personnalisés sont gérés via le **Node Material Editor** de BabylonJS et exportés en `.json`.

- **Formats Sources (Veille) :** Une analyse a été menée sur les formats sources optimaux (EXR/TIFF/TGA pour HDR vs PNG/WebP) et sur l'utilisation d'archives (comme ASAR ou ZIP) en tant que *conteneurs de sources* pour le pipeline de build, et non en tant qu'assets de runtime.

Partie 5 : Conclusion - Philosophie, Portée et Futur du Projet

5.1 Philosophie et Intérêt

Le projet "Élodie et Goupil" a évolué d'une simple idée de mod Quake à une architecture de jeu Web complète, modulaire et résiliente.
La veille technologique démontre une transition claire d'une dépendance à des technologies

héritées (Quake, PAK, WAD et UI codée en dur) vers une pile **100% open-source et "Web-native"** (glTF, KTX2, WebM, Web Workers).

La philosophie du projet est de :

1. **Prioriser la Performance :** Utilisation de formats GPU-natifs (KTX2) et isolation des threads (Web Workers).
2. **Garantir la Modularité :** Séparation stricte du "Moteur" (`engine/shared`) et du "Jeu" (`client/server`), permettant la réutilisation et la maintenance.
3. **Assurer la Résilience :** Conception "Offline-first" avec fallback automatique et stockage des intentions réseau.
4. **Pérenniser le Projet :** Utilisation exclusive de standards ouverts et libres de droits (Khronos, W3C) et anticipation des évolutions futures (SSR).

5.2 Résultat Actuel et Futur

Le projet dispose d'une architecture technique détaillée et d'un scénario narratif complet en 8 actes, prêt pour la phase de pré-production.
La pile technologique (BabylonJS + Workers + glTF/KTX2) est validée comme étant la solution optimale pour répondre aux ambitions narratives et techniques du projet.

La suite du projet soulève plusieurs questions stratégiques pour la prochaine phase de développement :

- **Implémentation du Rendu Côté Serveur (SSR) :** La structure est prête, mais quand et avec quelles technologies (WebGPU headless, WebRTC, streaming vidéo) cette fonctionnalité sera-t-elle implémentée pour les clients légers (mobiles, GPU non dédiés) ?
- **Optimisation des Builds Natifs :** Les dépôts `native` et `react-native` utiliseront-ils BabylonJS Native, ou des technologies tierces nécessitant une veille complémentaire ?
- **Gestion des Entités (Map-like) :** Comment la logique des `extras` glTF sera-t-elle gérée ? Un éditeur visuel (par exemple, un addon Blender personnalisé) sera-t-il développé pour faciliter la création de niveaux ?
- **Synchronisation Réseau :** Quelle stratégie de synchronisation réseau (snapshots complets, compression delta des états, ou "lockstep") sera utilisée pour le mode en ligne afin de gérer la latence de manière optimale ?